2.16 Markov Chain Monte Carlo Bayesian Learning for Neural Networks

Markov Chain Monte Carlo Bayesian Learning for Neural Networks

Michael S. Goodrich
Old Dominion University
mgood028@odu.edu

Abstract. Conventional training methods for neural networks involve starting at a random location in the solution space of the network weights, navigating an error hyper surface to reach a minimum, and sometime stochastic based techniques (e.g., genetic algorithms) to avoid entrapment in a local minimum. It is further typically necessary to preprocess the data (e.g., normalization) to keep the training algorithm on course. Conversely, Bayesian based learning is an epistemological approach concerned with formally updating the plausibility of competing candidate hypotheses thereby obtaining a posterior distribution for the network weights conditioned on the available data and a prior distribution. In this paper, we developed a powerful methodology for estimating the full residual uncertainty in network weights and therefore network predictions by using a modified Jeffery's prior combined with a Metropolis Markov Chain Monte Carlo method.

1.0 INTRODUCTION

We propose a methodology for estimating the full residual uncertainty in Artificial Neural Network (ANN) weights and therefore network predictions by using Bayesian probability analysis[4] (BPA), and a modified Jeffery's prior combined with computational sampling methods including Markov Chain Monte Carlo.

In this paper we restrict our attention to three layer feed-forward perceptrons, since they are sufficient[1,2] to serve as universal approximating functions. We further restrict attention to *supervised learning*. We will also not be considering feature extraction. As this effort is concerned with digital simulation based approaches, we will be using numerically driven discrete formulations (i.e. sums instead of integrals) throughout.

Artificial neural networks

An Artificial Neural Network can be thought of as a computational model which consists of three layers of processing units with full interconnection between layers such that each component of an input vector is scaled individually for each middle layer unit, and the scaled components are then summed and passed through a transfer or *activation* function in each unit in the middle layer and then the middle layer outputs constitute another vector as the input to the output layer which is likewise scaled independently and individually for each output unit. The units in the output layer are typically (but not necessarily) simple linear functions. The input vectors for each layer also contain an implicit component of 1 to serve as an input *bias*. Figure 1 depicts a conventional three-layer feed forward perceptron network.

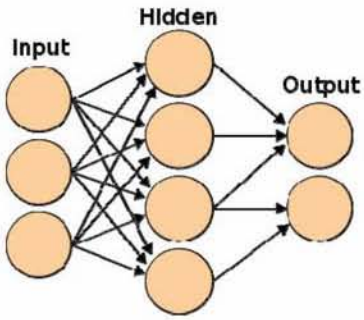


Figure 1 - Three Layer Feed Forward Network

Expressed as a mathematical model for the simplest case of a one input, one hidden unit, and one output we can write

$$y = v_0 + v_1 \psi(w_0 + w_1 x) \qquad (1)$$

where y is the output, $\vec{w}$ are the weights (scale factors) from the inputs $\vec{x}$ (which includes an implicit bias input of 1) to the hidden layer, $\vec{v}$ are the weights from the transfer function outputs (which includes an implicit bias input of 1) to the output layer,

and ψ is the non-linear activation function. Note the location of the bias components for v_0, and w_0.

For arbitrary numbers of inputs, hidden units, and outputs, equation (1) takes the form

$$y_k = v_{0k} + \sum_{h=1}^{Nh} v_{hk}\psi\left(w_{0h} + \sum_{i=1}^{Ni} w_{ih}x_i\right) \qquad (2)$$

and can be written as a matrix formulation.

.

Bayesian Probability

BPA requires determining a Universe of Discourse (UOD) which is a set of hypotheses that are ranked on a common scale of [0,1] in terms of their relative strength as an explanator of the observed data. This is done both for a family of competing models, and for competing sets of the parameter values for each model. The basic process is:

- determine a prior distribution for the model parameters of a given model
- determine a probabilistic likelihood function for the phenomena under study
- determine a UOD for our analysis
- determine a posterior distribution for the hypotheses in the UOD
- make inferences from the posterior yielding full accounting for the residual uncertainty of the parameters

This probability is then interpreted as a measure or weighting of the amount of inferential support[3] from the observed data for the hypotheses normed to entail the chosen UOD.

We wish to stress that the choices of prior, likelihood, and UOD represent *degrees of freedom* for the researcher; BPA only promises to give us the most logically justifiable results contingent on these choices[3,4].

Likelihood Model

The likelihood function is entirely dependent on the phenomena under study and must be constructed to yield the conditional probability of any observed data for a chosen model and values of its parameters.

Bayesian Prior Selection

Bayesian prior selection is a vast subject. Typically one may express ignorance concerning the current problem, or may possess some information that may be codified into a prior, e.g. by the Principle of Maximum Entropy[4,10,11]. In any case, the prior expresses our starting information concerning the parameters of the likelihood function.

Posterior Distribution

Bayesian posterior determination proceeds by computing Bayes Rule[4,13]

$$P(\vec{w} \mid D,M) = \frac{P_0(\vec{w} \mid M)L(D \mid \vec{w},M)}{\sum_{\{\vec{w}\}} P_0(\vec{w} \mid M)L(D \mid \vec{w},M)} \qquad (3)$$

where $P(\vec{w} \mid D,M)$ is the posterior probability of the model parameters $\vec{w}$ conditioned on the observed data D and the choice of model M, $P_0(\vec{w} \mid M)$ denotes the *prior* probability of the parameters $\vec{w}$ which summarizes all knowledge of $\vec{w}$ for this model brought forward into the present analysis, and $L(D \mid \vec{w},M)$ is the likelihood of the data being observed for the model given that the parameters have the value $\vec{w}$. The denominator of (3) is known as the *evidence* for the model:

$$P(D \mid M) = \sum_{\{\vec{w}\}} P_0(\vec{w} \mid M)L(D \mid \vec{w},M) \qquad (4)$$

and is the probability of the data given the model marginalized by the likelihood function over the hypothesis space $\{\vec{w}\}$.

Parameter Estimation

Bayesian parameter estimation proceeds by evaluating and scoring on a common probability scale, each value of the parameters in the UOD of model parameters using (3), thereby producing a normalized probability distribution (or mass function) over the UOD.

Model Selection

From (4) and using Bayes Rule (3) we can write

$$P(M \mid D) = \frac{P_0(M)P(D \mid M)}{\sum_{\{M'\}} P_0(M')P(D \mid M')} \qquad (5)$$

This posterior distribution over a separate UOD {M} *for models* allows selection of the model which best explains the observed data, again also yielding a full characterization of the residual uncertainty conditioned upon a choice of prior for the models and available data.

Occam Factors

BPA has an interesting feature where model selection is concerned in that it contains an explicit built in penalty for more complex models over simpler models. This feature is known as an Occam factor[11] and is a consequence of forming the ratio of the evidence for two competing models, one of more complexity than the other using equation (4). A factor that emerges in the ratio calculation will penalize[11] the more complex model due to its' greater expanse of parameter space that will be ultimately ruled out by conditioning on the available data.

Bayesian Predictions

Bayesian inference or prediction is generally concerned with the formal marginalization over the hypothesis space $\{\vec{w}\}$ of a given model. For example we might wish to check the probability of some desired output data $\vec{t}$ conditioned on our model, and our training data such that

$$P(\vec{t} \mid M, D) = \sum_{\{\vec{w}\}} P(\vec{t}, \vec{w} \mid M, D)$$

$$= \sum_{\{\vec{w}\}} P(\vec{t} \mid \vec{w}, M, D) P_0(\vec{w} \mid M) \qquad (6)$$

using the *product rule* of probability[4]. This is the predictive distribution for observing the data $\vec{t}$ formally treating the models parameters as *nuisance parameters*. For example if $P(\vec{t} \mid M, D)$ is materially different than the likelihood we might suspect our choice of likelihood function.

We might also form a simple *expectation* such that:

$$\langle \vec{y} \rangle = \sum_{\{\vec{w}\}} \vec{y}(\vec{w}) P(\vec{w} \mid D, M) \qquad (7)$$

where $P(\vec{w} \mid D, M)$ is given by (3), $\vec{y}(\vec{w})$ could be the output of an ANN with parameters $\vec{w}$, and the expectation is conditioned on the training data set D and choice of network represented by M.

Learning for Neural Networks

There remains the issue of determining the values of the weights $\vec{w}$ and $\vec{v}$ in (2) conditioned on the available data and any other relevant information. This is the central problem of ANN *learning*. We would like to point out that it is possible to determine the vector $\vec{v}$ as function of $\vec{y}$ and $\vec{w}$ with appropriate mathematical technique.

Backpropagation

The most common conventional (non Bayesian) approach to ANN learning is to concern oneself with an error function such as:

$$E = \frac{1}{N_p} \sum_{k=1}^{N_k} \sum_{p=1}^{N_p} \left[t_{p,k} - y_{p,k} \right]^2 \qquad (8)$$

As written, this is the mean squared error per pattern average for all outputs where N_k is the number of outputs, $t_{p,k}$ denotes the k^{th} desired output for the pth input pattern, $y_{p,k}$ denotes the k^{th} observed output for the p^{th}

input pattern, and N_p denotes the total number of observations or training data patterns.

The basic stratagem of backpropagation is to substitute (2) for the $y_{p,k}$ in (8), and then use error gradient information for the weights in order to use a numerical error reduction algorithm (typically *conjugate gradient*) to adjust the weights and achieve some error minimum.

Two outstanding issues emerge in this approach to learning:

- what is the proper minimum for the residual error that the network produces? This is the issue of *regularization* which is inhibition of network training to the noise component of the signal.

- What network model best explains the observed data?

Bayesian Learning for ANNs

In broad strokes, Bayesian Learning requires choosing a prior distribution over the network weights, framing a probabilistic formulation for an ANN model or models then using (2), (3), and (4) to determine the best network along with a posterior probability distribution over the weights for the selected model with a full characterization of the residual uncertainty in both. We describe our solution to these in section 2.

Monte Carlo Simulation

Because the Bayesian posterior - which is the sought after distribution - is a priori unknown, we must resort to some form of search strategy to find it. Monte Carlo simulations were originally developed to provide numerical integration of functions but can be used in a variety of ways to sample the solution space and determine the probability distribution for our chosen UOD, which we are choosing probabilistically as a Markov Chain rather than deterministically.

2.0 METHODOLGY

The principle challenge of our methodology is to combine Bayesian Probability, mathematical models of ANNs, and simulation based methods of solution search to determine a joint posterior probability distribution for the hidden network weights and any other parameters such as the noise or stochastic contribution to the observed data. Thus is created all that is necessary to make predictions with full accounting of the residual uncertainty in the inferred network.

Probabilistic Likelihood Model

We must determine a likelihood model to be used in equations (3),(4)

To address the issues of regularization (over fitting inhibition) and to account for actual residual stochasticity in the data, we choose to compose our "meta-model" as a linear combination of deterministic and non-deterministic or *stochastic* components. This requires expanding our hypothesis space to also ascertain the correct amount of stochasticity or loosely *noise* in the input data. To clarify, we seek to model the residual stochastic (loosely noise) component of the input data or signal as a form of regularization. To that end we model the likelihood of the target data (training pattern) less the output of the candidate network output $\vec{y}$ as:

$$L(\vec{t} - \vec{y} \mid \vec{w}, \Sigma) = G(|\vec{t} - \vec{y}(\vec{w})|, \Sigma) \qquad (9)$$

where $G(\blacksquare)$ is a Multivariate Gaussian probability density, $\vec{t}$ is the training pattern output data, and the components of $\vec{y}(\vec{w})$ are given by (2). That is to say, our likelihood model for the difference between the target (training) data and the candidate network output is to be modeled as a form

of *Multivariate Gaussian white noise*. Note that since white noise is uncorrelated, our likelihood model is *conditionally independent* between training inputs; we therefore model the stochastic part of the output as conditionally independent between training patterns while allowing for the possibility of correlations between the components of the output vector. We therefore write for the likelihood of the training set for a given choice of model parameters $\{\vec{w}, \Sigma\}$:

$$L(\vec{t} - \vec{y} \mid \vec{w}, \Sigma) = \prod_{p=1}^{N_p} G(\vec{t}_p - \vec{y}_p, \Sigma) \qquad (10)$$

Note that we have expanded our parameter search according to Bayesian principles to include the multivariate noise contribution Σ, the full covariance matrix for the difference between the observed output and modeled output. The full covariance matrix provides for possible correlations among the elements of the vector $\vec{t}_p - \vec{y}_p$ which is the modeled stochastic component associated with each training pattern vector $\vec{t}_p$. We use equation (10) for the likelihood function $L(\bullet)$ in (3) where the data D is now understood to be stochastic part of the training data i.e., $\vec{t} - \vec{y}(\vec{w})$ thus we are *absorbing* the deterministic part of the signal into the network output in such a manner as to maximize the probability associated with the stochastic *residue* of the training input via our likelihood function.

Using ANNs in this fashion can be thought of as a form of Bayesian *non parametric* probabilistic modeling with the choice of activation function serving as the appropriate basis functions[10]. (N.B.: the term "Non-Parametric Bayesian" has acquired a different meaning in the literature than what we are implying in this study)

Choice of Prior Distribution

Our methodology addresses the choice of a prior distribution by choosing a (modified) *Jeffreys'* prior[5,6,11] to express partial ignorance over the parameter space but also because it discriminates against excessively large model parameter values[12] (called *shrinkage* in the statistics literature, and *weight decay* in the ANN literature).

Jeffreys' prior is invariant to transformations of the parameter space and is related to the expected value of the Fisher Information Matrix. For scale parameters, this becomes

$$P(w \mid M) = c_0 / w \qquad (11)$$

where c_0 is a normalizing constant. This represents a density which is apportioned equally per decade of its scale and is therefore scale invariant. While the continuous version of this density is strictly improper (the cumulative distribution integral diverges), it is straightforward to construct a normalized discrete probability mass function over some chosen (always finite) UOD.

We consider the sought after network weights $\vec{w}$, and the noise contribution Σ to both be scale parameters. We modify the Jefferys' priors for both according to the following considerations:

- We impose a minimum value for each component of $\vec{w}$, and the diagonal components of Σ such that any values less than these cutoffs decay smoothly to zero
- We normalize the resulting discrete distributions over some reasonable range
- Since weight parameters $\vec{w}$ may be negative, we actually use the absolute value in (11), keeping the distribution in that case symmetric about the origin.

The resulting distributions have the general form of Figure 2 below.

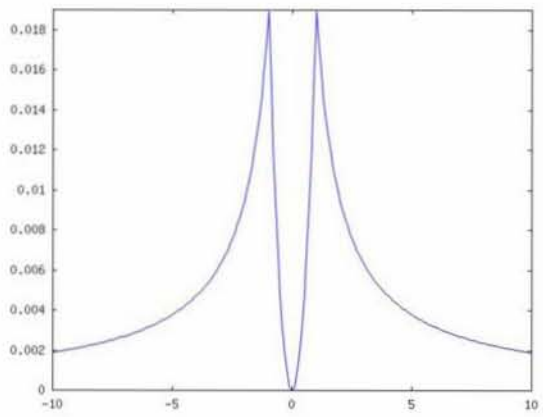


Figure 2 – Modified Jeffery's' prior density

For the diagonal components of the noise contribution parameter Σ we have only the positive (properly normalized) part of the curve. We do not discriminate against small values of the off-diagonal (covariance) elements of Σ

Strictly speaking, the cutoffs are somewhat arbitrary and good candidates for a hyperparameterized prior for each (in contemporary Bayesian fashion), but that is not included in this analysis. The actual choices made were such that the cutoff value were chosen sufficiently small to be hopefully good for a wide choice of problems.

These minimum are thought reasonable on the basis that network weights which are too small lead to uninteresting solutions, and if the noise contribution is too small then we are in effect eliminating that component of the modeling. In both cases, parameter values of 0 are clearly uninteresting.

MCMC

Choosing a UOD by Monte Carlo (MC) simulation tends to take one of two basic tracks:

1. Start at a random location and use a local rule to choose the next location
2. Use a global rule to choose the all locations

Conventional MCMC sampling techniques such as the Metropolis, Metropolis-Hastings, and Gibbs sampling are basically of type 1.

Independence Chain sampling, and Importance sampling are of type 2.

In our version of *grid or mesh sampling* we used a coarse grid to fine grid progression to characterize the posterior distribution and locate promising regions which were subsequently explored with a finer grid. This approach suffers from exponential increase in computational effort with increased dimensionality of the parameter space. The basic approach is to compute equation (3) for each grid point in the UOD, thus achieving a coarse grained posterior probability distribution. Finer grained computations over more promising regions then ensued. It is in this sense that a non-local or global rule is used in choosing sample points.

In a *random walk* oriented MCMC approach we used a Metropolis algorithm which generally is able to find promising regions, is less computationally demanding, but often may not give a complete characterization of the posterior distribution and in can yield lower quality network output when compared against the training data than grid sampling. The Metropolis algorithm operates by choosing its next point by constructing a Markov Chain via sampling from a *local proposal density* which is centered on the current point and for our study is a Multivariate Gaussian of the same dimension as the hypothesis space. The new point is then accepted with probability

$$P = \min\left\{1, \frac{P(\vec{w}_{new} \mid D, M)}{P(\vec{w}_{current} \mid D, M)}\right\} \qquad (12)$$

A chain of N points $\{\vec{w}\}$ is thus determined from this algorithm such that if a new candidate point is rejected, a copy of the current point is added to the chain. In this fashion, points are accumulated according

to their relative probability, the duplication of points serving to increase their respective weighting for inferences from the chain such as expectations according to

$$\langle \vec{y} \rangle = \frac{1}{N} \sum_{k=1}^{N} \vec{y}(\vec{w}_k) \qquad (13)$$

where $\vec{y}(\vec{w})$ is given from (2), and the inferential locus is the chain $\{\vec{w}\}$.

Thus the UOD in our methodology is determined stochastically by the sampling algorithm. Each sampled point accepted or rejected may be retained so as provide for a proper discrete probability distribution of the form

$$PD = \{\{\vec{w}_1, p_1\}, \ldots, \{\vec{w}_K, p_K\}\} \qquad (14)$$

for a distribution with K elements to be used in equations (6), and (7). Conventional MCMC doctrine uses (13). The proposal density used in (12) must be tuned in order to achieve acceptance rates of between 25% and 50% as recommended by the literature[14]. Typically MCMC sampling includes a *warm up time* to allow the chain to begin properly representing the target posterior distribution and so the warm up time is not included in equation (13).

3.0 EXPERIMENTS

General Remarks

We performed experiments on both simulation generated data and real world data. Simulation generated data is especially beneficial for validation of the methodology since the noise uncontaminated input data is readily available. Naturally, performing well in such a case gives one confidence in attacking real-world problems where the noise component may be unknown.

Noisy Sine Wave

This experiment is for a noisy sine wave and was pattern matched with a network consisting of 1 input, 2 hidden units, 1 output, with Gaussian noise and randomly sampled for 1000 trials

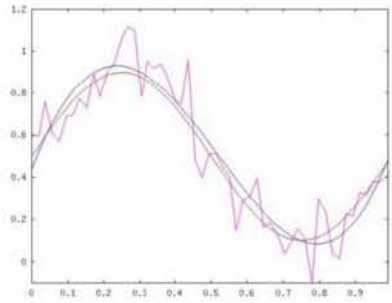


Figure 3 – Noisy Sine Wave

In Figure 3 the red sine wave is the true denoised signal, the noisy red line is the actual input, and the blue line is the prediction of the network.

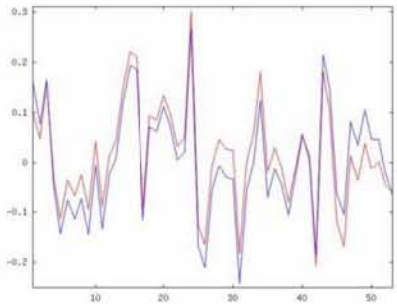


Figure 4 – Noise only (Sine Wave)

In Figure 4 the red curve is the true noise in the signal, and the blue line is the actual input less the prediction of the network, i.e, is the modeled noise resulting from the network prediction.

Decaying Exponential

This experiment is for a noisy decaying exponential curve and was pattern matched with a network consisting of 1 input, 1 hidden unit, 1 output, with Gaussian noise.

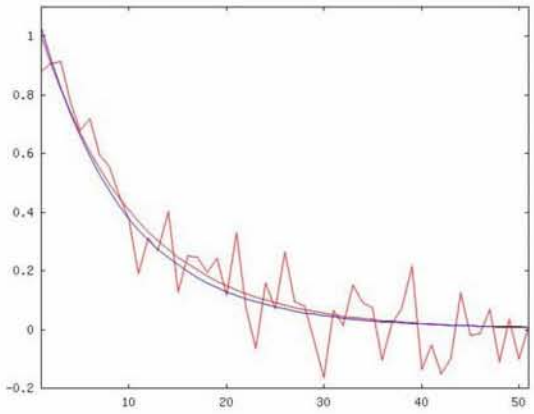


Figure 5 – Noisy Decaying Exponential

In Figure 5 the red line is the true denoised signal, the noisy red line is the actual input, and the blue line is the prediction of the network.

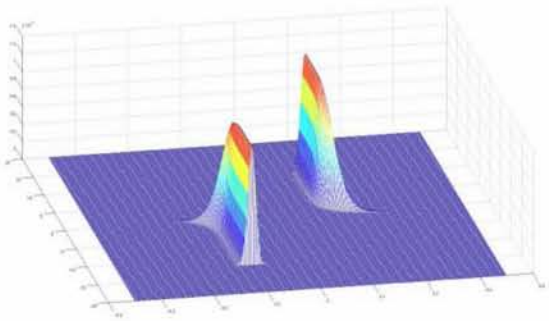


Figure 6 – Posterior for the Two Weights for Decaying Exponential

In Figure 6 we have the posterior distribution for the two hidden weights for the decaying exponential problem determined by grid sampling. There are clearly two branches of significant probability above some floor.

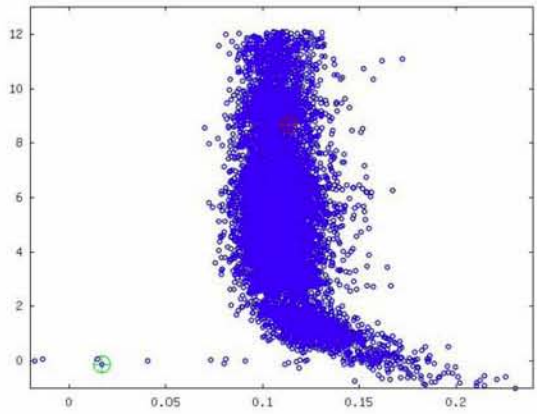


Figure 7 – Metropolis Markov Chain Samples for Two Weights for Decaying Exponential

In Figure 7 we have the sample points from the Metropolis algorithm for the two hidden weights for the decaying exponential problem. Note the correspondence between Figures 6 and 7. The Metropolis algorithm has found one branch of the solution depicted in Figure 6.

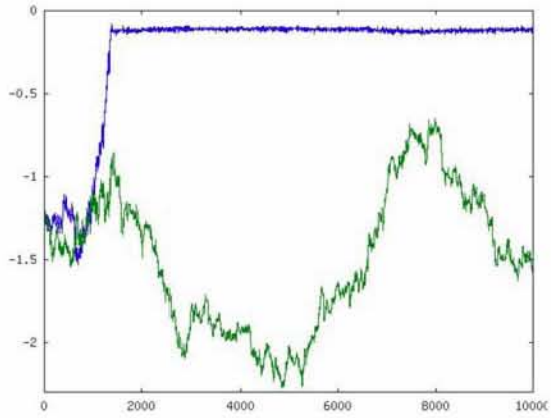


Figure 8 – Metropolis Markov Chain Process for Two Weights in Decaying Exponential

In Figure 8 we show the processes for the sample for the two weights of the decaying exponential fit. Note that the process for one of the parameters has found the correct value after a warm up of approximately 1500 steps in the Markov Chain. The other parameters' process is more of a random walk due to its wide range of acceptable values (compare with Figures 6 and 7).

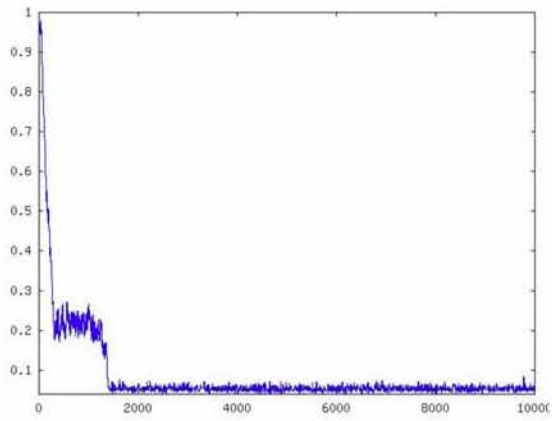


Figure 9 – Metropolis Markov Chain Process for Noise Parameter

In Figure 9 we show the process for the sample for the noise component parameter of the decaying exponential fit. We note that the process for the noise parameter has found the correct value of $\mathcal{N}(0,0.05^2)$ after a warm up of approximately 1500 steps in the Markov Chain.

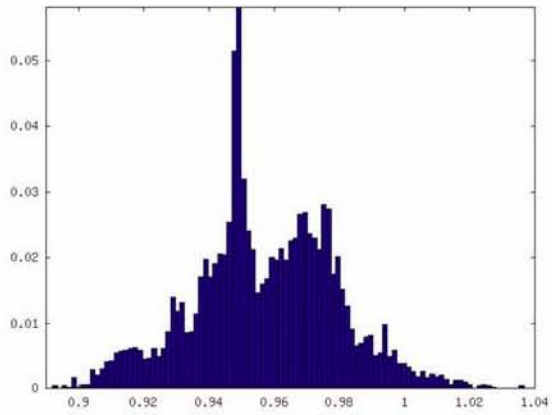


Figure 10 – Uncertainty in Network Prediction

In Figure 10 we show the normalized probability distribution from the resultant Markov Chain for the first output point. The denoised output for the first point is actually 1.

Concrete Problem

This problem known as the Concrete Problem is from the Machine Learning Database at UC Irvine and consists of 7 inputs and 3 outputs. It was processed with a network of 32 hidden units and only 500 MCMC samples after a warm up of 500 samples. Also included in the sampling for this problem were all the components of the full covariance matrix for the outputs including the off-diagonal components (ref. equations 9,10)

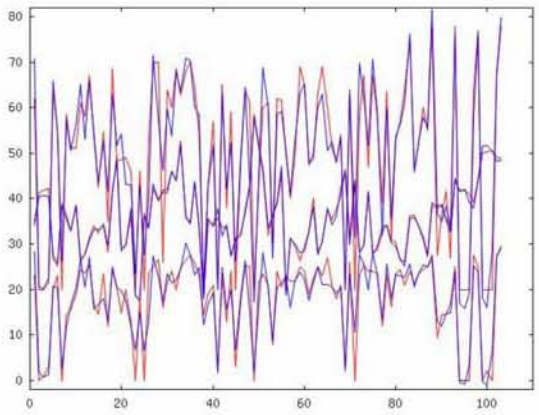


Figure 11 – Comparison between network output (blue) and training input (red) for 3 outputs in Concrete problem

In Figure 11 we compare the training data (red) with the network prediction (blue) for the concrete problem

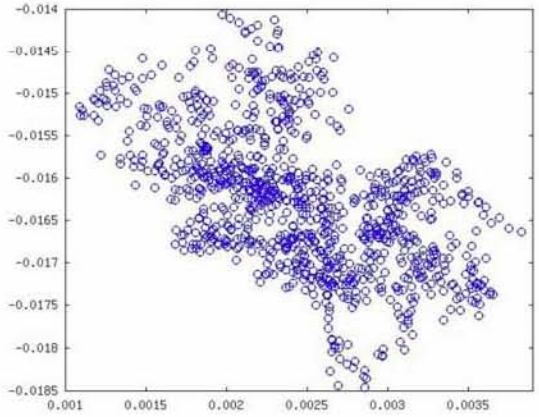


Figure 12 – Typical Metropolis Samples for Two of the Weights in Concrete problem

4.0 DISCUSSION / CONCLUSIONS

The selected experiments showed good responses of the methodology for surprisingly few numbers of trials. In cases where the actual strength of the noise component of the signal was known, the method reliably inferred a value very close to the actual value, with small variance on repeated trials. In all experiments we note the fairly rapid convergence of the chain to promising potential solutions starting from randomize initial locations. The implemented MCMC sampling algorithm was admittedly crude and typically achieved acceptance fractions of between 10% to 15% well below that recommended by the literature (25% to 50%).

5.0 REFERENCES

[1] G. Cybenko. Approximations by superpositions of sigmoidal functions. Mathematics of Control, Signals, and Systems, 2: no. 4 pp. 303-314. 1989.

[2] K. Hornik: Approximation Capabilities of Multilayer Feedforward Networks. Neural Networks, vol. 4, 1991.

[3] R. T. Cox, The Algebra of Probable Inference. The Johns Hopkins Press, 1961

[4] E. T. Jaynes, Probability Theory: The Logic of Science, Cambridge University Press., 2003

[5] Jeffreys, H An Invariant Form for the Prior Probability in Estimation Problems Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, Vol. 186, No. 1007 (Sep. 24, 1946), pp. 453-461

[6] Jeffreys, H.. Theory of Probability, third edition, Oxford University Press, 1961

[7] Roberts, G.O.; Gelman, A.; Gilks, W.R. (1997). "Weak convergence and optimal scaling of random walk Metropolis algorithms", *Ann. Appl. Probab.* 7 (1): 110–120

[8] Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. (1953). "Equations of State Calculations by Fast Computing Machines", *Journal of Chemical Physics* 21 (6): 1087–1092

[9] Hastings, W.K. (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications", *Biometrika* **57** (1): 97–109

[10] Sivia, D.S. "Data Analysis: A Bayesian Tutorial". *Oxford University Press,* 1996, ch 6.

[11] Gregory, P. "Bayesian Logical Data Analysis for the Physical Sciences", *Cambidge University Press,* 2005, ch 3.

[12] Bishop, C.M. "Pattern Recognition and Machine Learning", *Springer Press,* 2006, p. 10

[13] Lee, P. "Bayesian Statistics: An Intorduction", *Oxford University Press,* 1989, pp 90-97.

[14] Gamerman, D., Lopes, H., "Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference", 2nd Ed., , *Chapman & Hall/CRC,* 2006.

6.0 ACKNOWLEDGMENT(S)

We would like to thank Dr. Jiang Li, Old Dominion University, for being supportive of this line of research, and for suggesting that a paper along these lines be presented at this conference.

Markov Chain Monte Carlo Bayesian Learning for Neural Networks

presented by

Michael S. Goodrich
Modelling and Simulation Ph. D. Program,
Old Dominion University,
Alion Sciences and Technology

Introduction

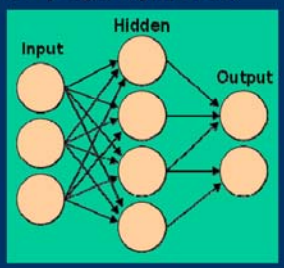
- I am a Modeling and Simulation Ph.D. student at ODU and an employee of Alion Science and Technology currently researching issues in computation Bayesian probability analysis in the context of problems in Machine Learning.
- My research thrust consists of a combination of
  - Bayesian Model Testing
  - Bayesian Parameter Estimation
  - Adaptive Monte Carlo sampling
  - Information Theoretic Probabilistic analysis
  - ANN Constraint Analysis
  - Probabilistic Inference

Outline

- ANNs
- BPA
- ANN (Machine) Learning
- MC simulation
- Markov Chains
- MCMC
- Methodology
  - Likelihood Modeling
  - Prior Distribution for Weights
  - MCMC Sampling Technique
  - MCMC Proposal Distribution
- Experiments
  - Exponential Curve
  - Sine curve
  - Concrete Problem
- Discussion/Conclusions

Neural Networks

- *Regression – Determines a regression curve to sample data*
- *Classification – Maps a non-linear decision boundary to a linear decision boundary in the feature space of the non-linear basis functions (e.g. sigmoids)*



$$y_k = v_{0k} + \sum_{h=1}^{Nh} v_{hk} \psi \left(w_{0h} + \sum_{i=1}^{Ni} w_{ih} x_i \right)$$

Supervised Learning Problem

- Given: Training examples $\{(\vec{x}, \vec{y})\}$
- Find some function f s.t., $\vec{y} = f(\vec{x})$

Gradient Descent

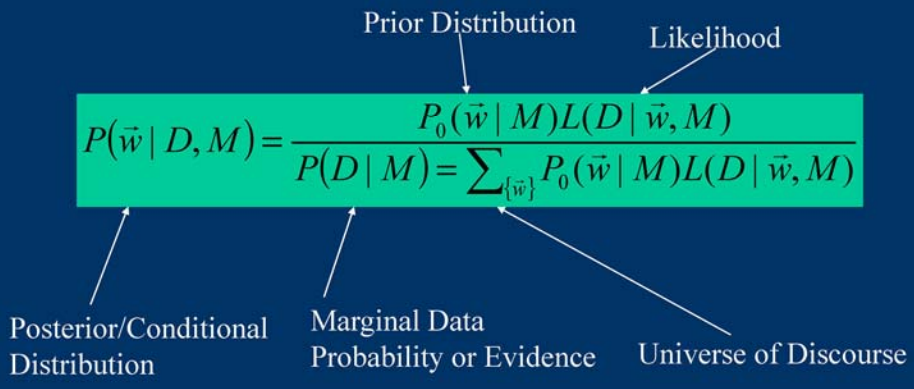
Gradient descent: $\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla E(\mathbf{w})$

The error function (LMSE) $E = \frac{1}{N_p} \sum_{k=1}^{N_k} \sum_{p=1}^{N_p} \left[t_{p,k} - y_{p,k}(\vec{w}) \right]^2$

Problem! *Not* LMSE -> Regularization: Theoretical basis?

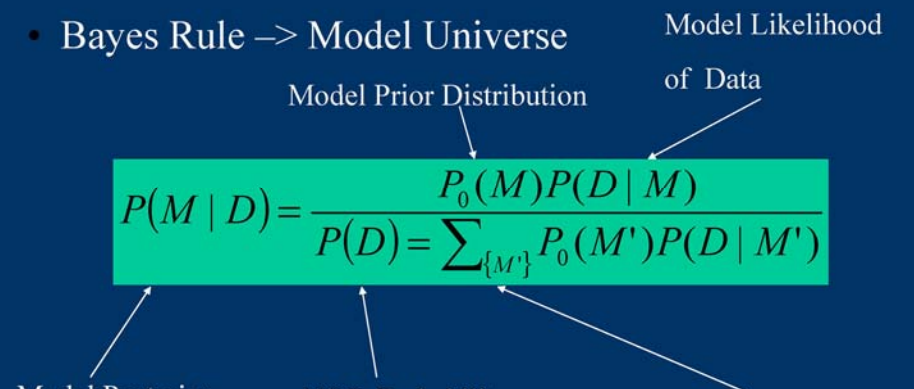
Bayesian Statistics in a Nutshell

- Bayes Rule: Natural Learning Law from Data



Bayesian Statistics in a Nutshell

- Bayes Rule –> Model Universe



Bayesian Marginalization

Bayesian prediction/inference is usually difficult and/or expensive -> requires *Marginalization*

Conditional Inference (Distribution)

$$p(w_1 \mid D) = \sum_{w_2,\ldots,w_n} p(w_1, w_2, \ldots, w_n \mid D)$$

$$P(d \mid D, M) = \sum_{\{\vec{w}\}} P(d, \vec{w} \mid D, M) = \sum_{\{\vec{w}\}} P(\vec{w} \mid D, M) L(d \mid \vec{w}, M)$$

Predictive Distribution

Posterior/Conditional Distribution

Likelihood

Bayesian Probability: Researcher Degrees of Freedom

- Initial Conditions: Prior Distributions
- Phenomena: Likelihood function
- Purpose: Inferences / Predictions

Discrete Markov Chains

- A sampling sequence {x} from a *proposal distribution* forms a ***Markov chain*** if it has the property: $p_p(x_n \mid x_{n-1},\ldots,x_1) = p_p(x_n \mid x_{n-1})$
- *Markov Chain Monte Carlo* is a chain construction technique for converging a chain to some *target distribution* p_t which is unknown in advance, s.t.:

$$p_t(x) = \lim_{N\to\infty}\{n_x, x\} \leftarrow p_p(x_n \mid x_{n-1})$$

- Thus

$$\langle f(x)\rangle = \lim_{N\to\infty}\sum_{n}^{N_s} p_t(x_n) f(x_n) \approx \frac{1}{N_s}\sum_{n}^{N_s} f(x_n)$$

Metropolis Sampling

- Constructs a Markov chain by *proposing* the next point in the chain s.t.: $\vec{w}_{new} \leftarrow f(\vec{w}_{current}, \underline{\underline{\Sigma}}) \sim \vec{w}_{current} + N(0, \underline{\underline{\Sigma}})$
- *Accepting* the proposed next point with probability:

$$P = \min\left\{1, \frac{P(\vec{w}_{new} \mid D, M)}{P(\vec{w}_{current} \mid D, M)}\right\}$$

- Where

$$P(\vec{w} \mid D, M) \propto L(D \mid \vec{w}, M)\pi(\vec{w} \mid M)$$

Methodology

- Likelihood function
- Prior density for network weights
- MC Sampling Technique
- Predictions / Inferences

Neural Networks Likelihood Function

- Likelihood of training set for model choice:

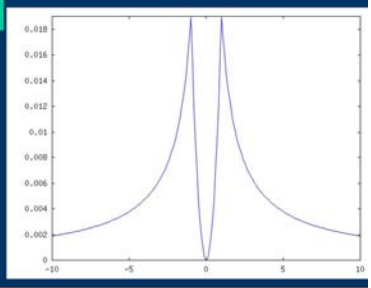
$$L(\vec{t} - \vec{y} \mid \vec{w}, \Sigma) = N(|\vec{t} - \vec{y}(\vec{w})|, \Sigma)$$

- White noise is *uncorrelated* so L() factorizes as:

$$L(\vec{t} - \vec{y} \mid \vec{w}, \Sigma) = \prod_{p=1}^{N_p} N(\vec{t}_p - \vec{y}_p, \Sigma)$$

Modified Jeffreys' Prior

- **Jeffreys':** Concerned with specifying transformation invariant priors to represent *ignorance*.
- Must address both *location* and *scale* parameters.
- Must define ignorance by a specific *transformation* (Jaynes).
- Consider $\mu' = \mu + a_0; \sigma' = b_0\sigma$ s.t. $f(\mu,\sigma) \Leftrightarrow g(\mu',\sigma')$
- General Solution is: $p(\mu,\sigma) = const \times \frac{1}{\sigma}$



- ANN -> $\vec{w}$; Noise-> $N(0,\sigma^2)$ *scale* parameters
- Modify to discriminate against near zero values:

MC Sampling Technique

Initial Proposal Distribution $N(\vec{w}_{new} - \vec{w}_{old}, mcmcsd_0^{\,2})$

Scaling Rules

if(acceptance fraction < 25%) -> increase mcmcsd

if(acceptance fraction > 50%) -> decrease mcmcsd

Inferences / Predictions

Expectations $\langle f(x) \rangle = \lim_{N \to \infty} \sum_{n}^{N_s} p_t(x_n) f_n(x_n) \approx \frac{1}{N_s} \sum_{n}^{N_s} f_n(x_n)$

Posterior Distributions: $histogram\left(\{f_n(\vec{w}_n)\}\right)$

E.g.,

- Network Output
- Weights
- Noise Parameter
- Vector Noise Covariance Matrix

Inferences / Predictions

Expected Network Output: $\langle y(x \mid \vec{w}) \rangle = \lim_{N \to \infty} \sum_{n}^{N_s} p_t(w_n) y_n(x \mid \vec{w}_n) \approx \frac{1}{N_s} \sum_{n}^{N_s} y_n(x \mid \vec{w}_n)$

Network Output Stats: $histogram\left(\{y_n(x \mid \vec{w}_n)\}\right)$

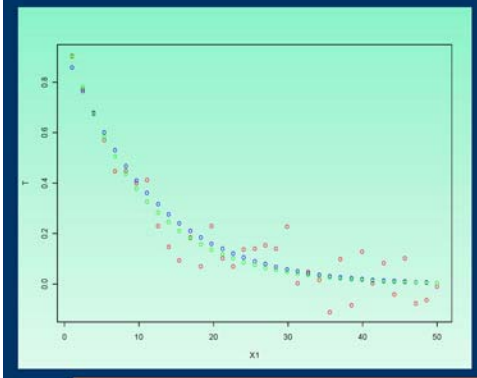
Weight Distribution: $\langle \vec{w} \rangle = \lim_{N \to \infty} \sum_{n}^{N_s} p_t(w_n) \vec{w}_n \approx \frac{1}{N_s} \sum_{n}^{N_s} \vec{w}_n$

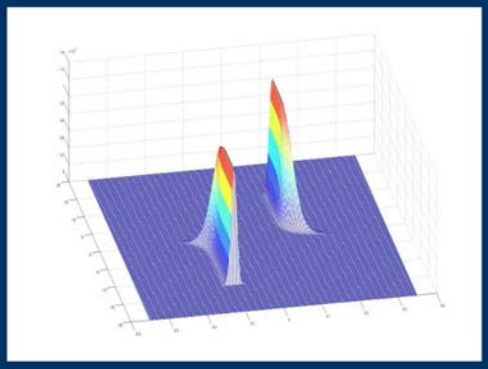
Noise Distribution: $\langle \sigma \rangle = \lim_{N \to \infty} \sum_{n}^{N_s} p_t(w_n) \sigma_n \approx \frac{1}{N_s} \sum_{n}^{N_s} \sigma_n$

(Vector) Noise Distribution: $\langle \Sigma \rangle = \lim_{N \to \infty} \sum_{n}^{N_s} p_t(w_n) \Sigma_n \approx \frac{1}{N_s} \sum_{n}^{N_s} \Sigma_n$

Noisy Exponential Curve

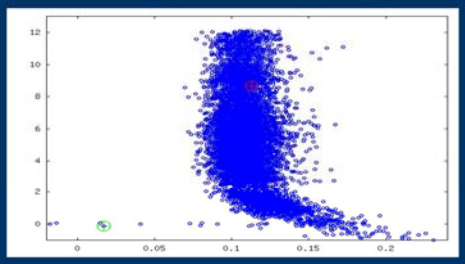
- Simple network of 1 input, 1 hidden unit, 1 output
- Solution state space is two weights
- Actually data is $y = \exp\left(\frac{-x}{10}\right) + N(0,0.05^2) = v_0 + v_1\psi(w_0 + w_1 x)$

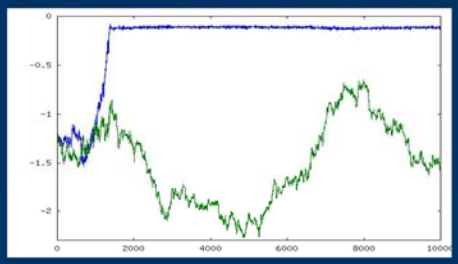


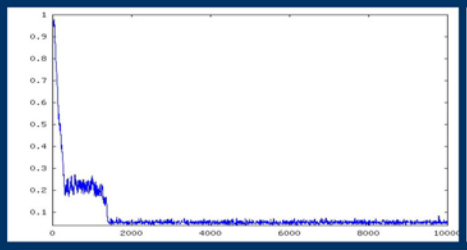


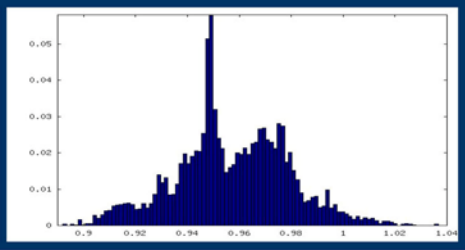
Noisy Exponential Curve

- Metropolis sampling with proposal = $N\left(\vec{w}_{new} - \vec{w}_{old}, \left(10^{-2}\right)^2\right)$



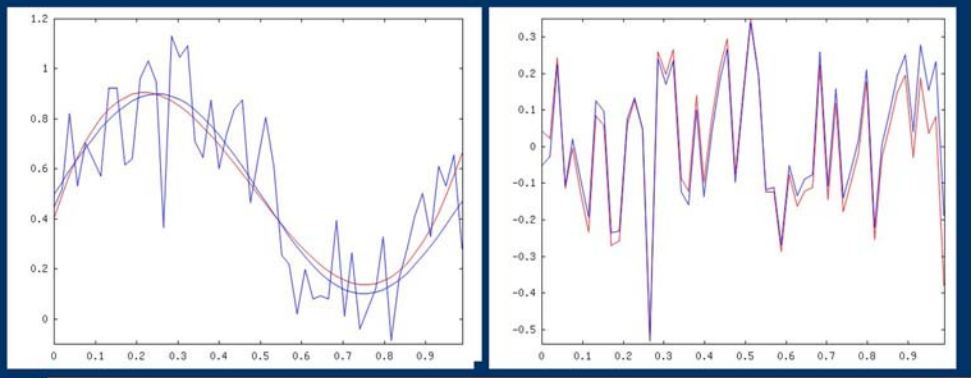






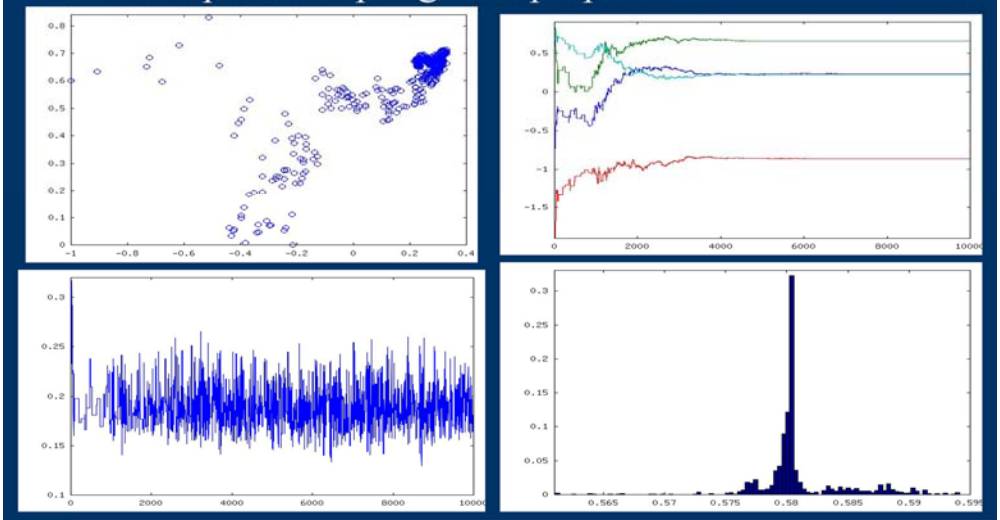
Noisy Sine Curve

- Network of 1 input, 2 hidden unit, 1 output
- Solution state space is four weights
- Actually data is $y = \frac{1}{2} + 0.4\sin(2\pi x) + N(0,0.2^2) = v_0 + v_1\psi(w_3 + w_0 x) + v_2\psi(w_4 + w_4 x)$



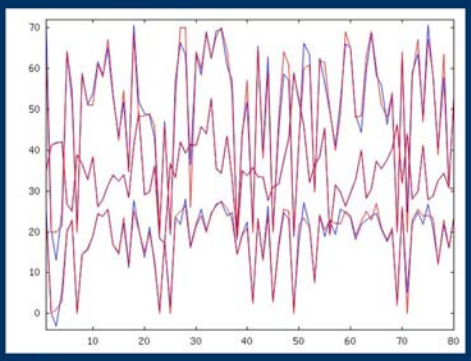
Noisy Sine Curve

- Metropolis sampling with proposal = $N\left(\vec{w}_{new} - \vec{w}_{old}, \left(10^{-2}\right)^2\right)$



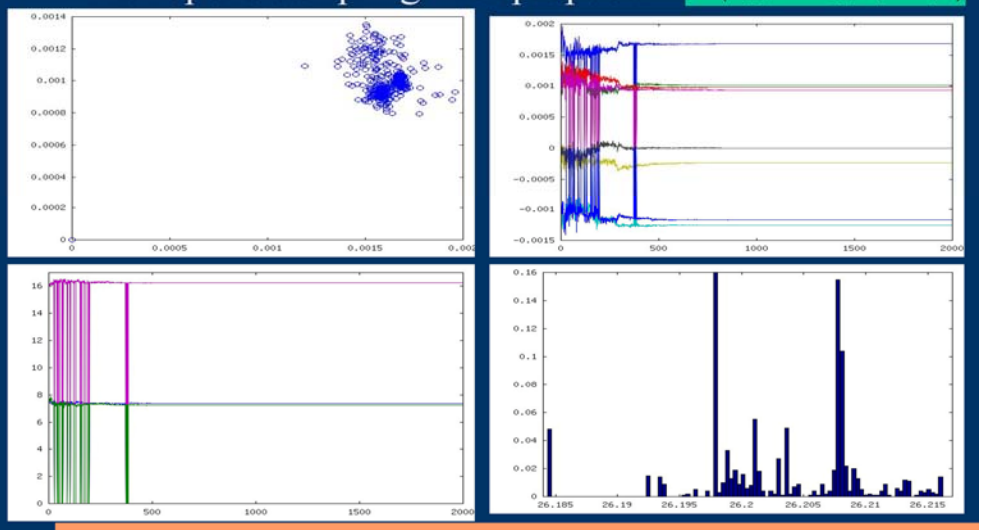
Concrete Problem

- Network of 7 inputs, 64 hidden units, 3 outputs
- Solution state space is (7+1)64 = 512 weights
- Actually data is real-world -> "noise" component unknown



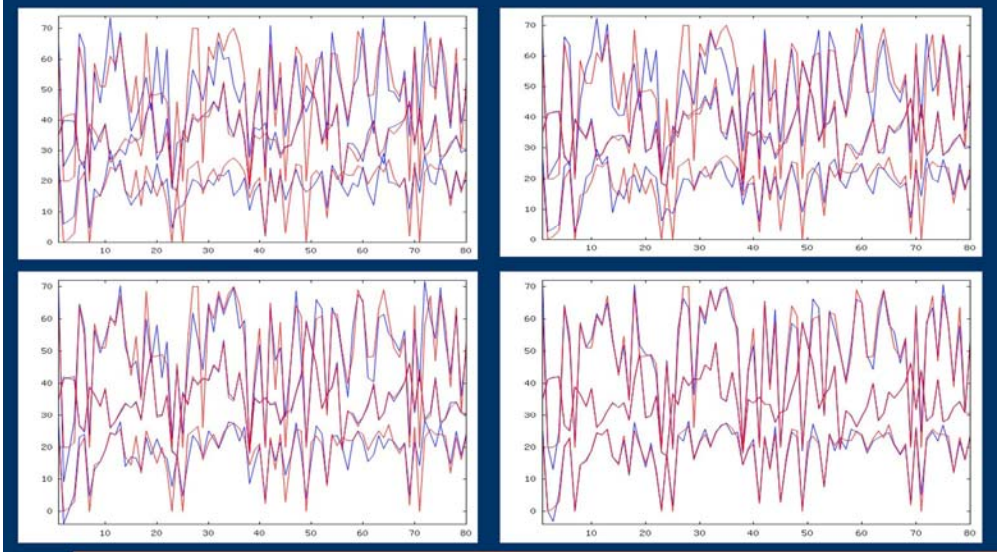
Concrete Problem

- Metropolis sampling with proposal = $N\left(\vec{w}_{new} - \vec{w}_{old}, \left(10^{-3}\right)^2\right)$



Concrete Problem

- Comparative Predictions of 8,16,32,64 hidden units



Discussion / Conclusions

- A promising start!
- Surprisingly good results for small sample sizes
- Other options for priors:
  - Constraints
  - Non Linear solvers
- Proposal Density Scaling Issues
- Regularization: Model Selection appears best.